# Python: module browser.gui_annotate

## browser.gui_annotate

index

```
# The PCMDI Data Browser Plot Annotation -  gui_annotate module
#
################################################################################
#                                                                              #
# Module:       gui_annotate module                                            #
#                                                                              #
# Copyright:    "See file Legal.htm for copyright information."               #
#                                                                              #
# Authors:      PCMDI Software Team                                            #
#               Lawrence Livermore National Laboratory:                        #
#               support@pcmdi.llnl.gov                                         #
#                                                                              #
# Description:  PCMDI Software System browser VCS plot annotation.            #
#                                                                              #
# Version:      4.0                                                            #
#                                                                              #
################################################################################
#
#---------------------------------------------------------------------
# NOTE: need to use version of Python that imports Tkinter and Pmw
#---------------------------------------------------------------------
```

## *Modules*

| | | | |
|---|---|---|---|
| Numeric | cdtime | browser.gui_menu | sys |
| Tkinter | gui_support.gui_color | browser.gui_set_text_object | vcs |
| __main__ | browser.gui_control | math | browser.vcs_function |
| cdms | browser.gui_functions | string | |

## *Classes*

Tkinter.Button(Tkinter.Widget)
    EntryButton
create

class ***EntryButton***(Tkinter.Button)

Method resolution order:
    EntryButton
    Tkinter.Button

Tkinter.Widget
Tkinter.BaseWidget
Tkinter.Misc
Tkinter.Pack
Tkinter.Place
Tkinter.Grid

---

Methods defined here:

***__init__***(self, master=None, cnf={}, **kw)

---

Methods inherited from Tkinter.Button:

***flash***(self)
```
    Flash the button.

    This is accomplished by redisplaying
    the button several times, alternating between active and
    normal colors. At the end of the flash the button is left
    in the same normal/active state as when the command was
    invoked. This command is ignored if the button's state is
    disabled.
```

***invoke***(self)
```
    Invoke the command associated with the button.

    The return value is the return value from the command,
    or an empty string if there is no command associated with
    the button. This command is ignored if the button's state
    is disabled.
```

***tkButtonDown***(self, *dummy)

***tkButtonEnter***(self, *dummy)

***tkButtonInvoke***(self, *dummy)

***tkButtonLeave***(self, *dummy)

***tkButtonUp***(self, *dummy)

---

Methods inherited from Tkinter.BaseWidget:

***destroy***(self)
```
    Destroy this and all descendants widgets.
```

---

Methods inherited from Tkinter.Misc:

***__getitem__*** = cget(self, key)
```
    Return the resource value for a KEY given as string.
```

***__setitem__***(self, key, value)

***__str__***(self)
```
Return the window path name of this widget.
```

***after***(self, ms, func=None, *args)
```
Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the
function which shall be called. Additional parameters
are given as parameters to the function call.  Return
identifier to cancel scheduling with after_cancel.
```

***after_cancel***(self, id)
```
Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be
given as first parameter.
```

***after_idle***(self, func, *args)
```
Call FUNC once if the Tcl main loop has no event to
process.

Return an identifier to cancel the scheduling with
after_cancel.
```

***bbox*** = grid_bbox(self, column=None, row=None, col2=None, row2=None)
```
Return a tuple of integer coordinates for the bounding
box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from
the cell with row and column 0 to the specified
cell. If COL2 and ROW2 are given the bounding box
starts at that cell.

The returned integers specify the offset of the upper left
corner in the master widget and the width and height.
```

***bell***(self, displayof=0)
```
Ring a display's bell.
```

***bind***(self, sequence=None, func=None, add=None)
```
Bind to this widget at event SEQUENCE a call to function FUNC

SEQUENCE is a string of concatenated event
patterns. An event pattern is of the form
<MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one
of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4,
Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3,
B3, Alt, Button4, B4, Double, Button5, B5 Triple,
Mod1, M1. TYPE is one of Activate, Enter, Map,
ButtonPress, Button, Expose, Motion, ButtonRelease
```

```
FocusIn, MouseWheel, Circulate, FocusOut, Property,
Colormap, Gravity Reparent, Configure, KeyPress, Key,
Unmap, Deactivate, KeyRelease Visibility, Destroy,
Leave and DETAIL is the button number for ButtonPress,
ButtonRelease and DETAIL is the Keysym for KeyPress and
KeyRelease. Examples are
<Control-Button-1> for pressing Control and mouse button 1 or
<Alt-A> for pressing A and the Alt key (KeyPress can be omitt
An event pattern can also be a virtual event of the form
<<AString>> where AString can be arbitrary. This
event can be generated by event_generate.
If events are concatenated they must appear shortly
after each other.

FUNC will be called if the event sequence occurs with an
instance of Event as argument. If the return value of FUNC is
"break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC wi
be called additionally to the other bound function or whether
it will replace the previous function.

Bind will return an identifier to allow deletion of the bound
unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list
of bound events are returned.
```

***bind_all***(self, sequence=None, func=None, add=None)

```
Bind to all widgets at an event SEQUENCE a call to function F
An additional boolean parameter ADD specifies whether FUNC wi
be called additionally to the other bound function or whether
it will replace the previous function. See bind for the retur
```

***bind_class***(self, className, sequence=None, func=None, add=None)

```
Bind to widgets with bindtag CLASSNAME at event
SEQUENCE a call of function FUNC. An additional
boolean parameter ADD specifies whether FUNC will be
called additionally to the other bound function or
whether it will replace the previous function. See bind for
the return value.
```

***bindtags***(self, tagList=None)

```
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated w
this widget. With a list of strings as argument the bindtags
set to this list. The bindtags determine in which order event
processed (see bind).
```

***cget***(self, key)

```
Return the resource value for a KEY given as string.
```

***clipboard_append***(self, string, **kw)
```
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword
argument specifies the target display. The clipboard
can be retrieved with selection_get.
```

***clipboard_clear***(self, **kw)
```
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword
argument specifies the target display.
```

***colormodel***(self, value=None)
```
Useless. Not implemented in Tk.
```

***columnconfigure*** = grid_columnconfigure(self, index, cnf={}, **kw)
```
Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column),
weight (how much does additional space propagate to this colu
and pad (how much space to let additionally).
```

***config*** = configure(self, cnf=None, **kw)
```
Configure resources of a widget.

The values for resources are specified as keyword
arguments. To get an overview about
the allowed keyword arguments call the method keys.
```

***configure***(self, cnf=None, **kw)
```
Configure resources of a widget.

The values for resources are specified as keyword
arguments. To get an overview about
the allowed keyword arguments call the method keys.
```

***deletecommand***(self, name)
```
Internal function.

Delete the Tcl command provided in NAME.
```

***event_add***(self, virtual, *sequences)
```
Bind a virtual event VIRTUAL (of the form <<Name>>)
to an event SEQUENCE such that the virtual event is triggered
whenever SEQUENCE occurs.
```

***event_delete***(self, virtual, *sequences)
```
Unbind a virtual event VIRTUAL from SEQUENCE.
```

***event_generate***(self, sequence, **kw)

```
Generate an event SEQUENCE. Additional
keyword arguments specify parameter of the event
(e.g. x, y, rootx, rooty).
```

***event_info***(self, virtual=None)

```
Return a list of all virtual events or the information
about the SEQUENCE bound to the virtual event VIRTUAL.
```

***focus*** = focus_set(self)

```
Direct input focus to this widget.

If the application currently does not have the focus
this widget will get the focus if the application gets
the focus through the window manager.
```

***focus_displayof***(self)

```
Return the widget which has currently the focus on the
display where this widget is located.

Return None if the application does not have the focus.
```

***focus_force***(self)

```
Direct input focus to this widget even if the
application does not have the focus. Use with
caution!
```

***focus_get***(self)

```
Return the widget which has currently the focus in the
application.

Use focus_displayof to allow working with several
displays. Return None if application does not have
the focus.
```

***focus_lastfor***(self)

```
Return the widget which would have the focus if top level
for this widget gets the focus from the window manager.
```

***focus_set***(self)

```
Direct input focus to this widget.

If the application currently does not have the focus
this widget will get the focus if the application gets
the focus through the window manager.
```

***getboolean***(self, s)

```
Return a boolean value for Tcl boolean values true and false
```

***getvar***(self, name='PY_VAR')

```
Return value of Tcl variable NAME.
```

***grab_current***(self)

```
    Return widget which has currently the grab in this applicatio
    or None.
```

***grab_release***(self)

```
    Release grab for this widget if currently set.
```

***grab_set***(self)

```
    Set grab for this widget.

    A grab directs all events to this and descendant
    widgets in the application.
```

***grab_set_global***(self)

```
    Set global grab for this widget.

    A global grab directs all events to this and
    descendant widgets on the display. Use with caution -
    other applications do not get events anymore.
```

***grab_status***(self)

```
    Return None, "local" or "global" if this widget has
    no, a local or a global grab.
```

***grid_bbox***(self, column=None, row=None, col2=None, row2=None)

```
    Return a tuple of integer coordinates for the bounding
    box of this widget controlled by the geometry manager grid.

    If COLUMN, ROW is given the bounding box applies from
    the cell with row and column 0 to the specified
    cell. If COL2 and ROW2 are given the bounding box
    starts at that cell.

    The returned integers specify the offset of the upper left
    corner in the master widget and the width and height.
```

***grid_columnconfigure***(self, index, cnf={}, **kw)

```
    Configure column INDEX of a grid.

    Valid resources are minsize (minimum size of the column),
    weight (how much does additional space propagate to this colu
    and pad (how much space to let additionally).
```

***grid_location***(self, x, y)

```
    Return a tuple of column and row which identify the cell
    at which the pixel at position X and Y inside the master
    widget is located.
```

***grid_propagate***(self, flag=['_noarg_'])

```
    Set or get the status for propagation of geometry informatior

    A boolean argument specifies whether the geometry informatior
    of the slaves will determine the size of this widget. If no a
```

```
is given, the current setting will be returned.
```

***grid_rowconfigure***(self, index, cnf={}, **kw)

```
Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row),
weight (how much does additional space propagate to this row)
and pad (how much space to let additionally).
```

***grid_size***(self)

```
Return a tuple of the number of column and rows in the grid.
```

***grid_slaves***(self, row=None, column=None)

```
Return a list of all slaves of this widget
in its packing order.
```

***image_names***(self)

```
Return a list of all existing image names.
```

***image_types***(self)

```
Return a list of all available image types (e.g. phote bitmap
```

***keys***(self)

```
Return a list of all resource names of this widget.
```

***lift*** = tkraise(self, aboveThis=None)

```
Raise this widget in the stacking order.
```

***lower***(self, belowThis=None)

```
Lower this widget in the stacking order.
```

***mainloop***(self, n=0)

```
Call the mainloop of Tk.
```

***nametowidget***(self, name)

```
Return the Tkinter instance of a widget identified by
its Tcl name NAME.
```

***option_add***(self, pattern, value, priority=None)

```
Set a VALUE (second parameter) for an option
PATTERN (first parameter).

An optional third parameter gives the numeric priority
(defaults to 80).
```

***option_clear***(self)

```
Clear the option database.

It will be reloaded if option_add is called.
```

***option_get***(self, name, className)

```
Return the value for an option NAME for this widget
with CLASSNAME.

Values with higher priority override lower values.
```

***option_readfile***(self, fileName, priority=None)

```
Read file FILENAME into the option database.

An optional second parameter gives the numeric
priority.
```

***pack_propagate***(self, flag=['_noarg_'])

```
Set or get the status for propagation of geometry informatio

A boolean argument specifies whether the geometry informatio
of the slaves will determine the size of this widget. If no a
is given the current setting will be returned.
```

***pack_slaves***(self)

```
Return a list of all slaves of this widget
in its packing order.
```

***place_slaves***(self)

```
Return a list of all slaves of this widget
in its packing order.
```

***propagate*** = pack_propagate(self, flag=['_noarg_'])

```
Set or get the status for propagation of geometry informatio

A boolean argument specifies whether the geometry informatio
of the slaves will determine the size of this widget. If no a
is given the current setting will be returned.
```

***quit***(self)

```
Quit the Tcl interpreter. All widgets will be destroyed.
```

***register*** = _register(self, func, subst=None, needcleanup=1)

```
Return a newly created Tcl function. If this
function is called, the Python function FUNC will
be executed. An optional function SUBST can
be given which will be executed before FUNC.
```

***rowconfigure*** = grid_rowconfigure(self, index, cnf={}, **kw)

```
Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row),
weight (how much does additional space propagate to this row)
and pad (how much space to let additionally).
```

***selection_clear***(self, **kw)

```
Clear the current X selection.
```

***selection_get***(self, **kw)
```
Return the contents of the current X selection.

A keyword parameter selection specifies the name of
the selection and defaults to PRIMARY.  A keyword
parameter displayof specifies a widget on the display
to use.
```

***selection_handle***(self, command, **kw)
```
Specify a function COMMAND to call if the X
selection owned by this widget is queried by another
application.

This function must return the contents of the
selection. The function will be called with the
arguments OFFSET and LENGTH which allows the chunking
of very long selections. The following keyword
parameters can be provided:
selection - name of the selection (default PRIMARY),
type - type of the selection (e.g. STRING, FILE_NAME).
```

***selection_own***(self, **kw)
```
Become owner of X selection.

A keyword parameter selection specifies the name of
the selection (default PRIMARY).
```

***selection_own_get***(self, **kw)
```
Return owner of X selection.

The following keyword parameter can
be provided:
selection - name of the selection (default PRIMARY),
type - type of the selection (e.g. STRING, FILE_NAME).
```

***send***(self, interp, cmd, *args)
```
Send Tcl command CMD to different interpreter INTERP to be ex
```

***setvar***(self, name='PY_VAR', value='1')
```
Set Tcl variable NAME to VALUE.
```

***size*** = grid_size(self)
```
Return a tuple of the number of column and rows in the grid.
```

***slaves*** = pack_slaves(self)
```
Return a list of all slaves of this widget
in its packing order.
```

***tk_bisque***(self)
```
Change the color scheme to light brown as used in Tk 3.6 and
```

***tk_focusFollowsMouse***(self)

```
The widget under mouse will get automatically focus. Can not
be disabled easily.
```

***tk_focusNext***(self)

```
Return the next widget in the focus order which follows
widget which has currently the focus.

The focus order first goes to the next child, then to
the children of the child recursively and then to the
next sibling which is higher in the stacking order.  A
widget is omitted if it has the takefocus resource set
to 0.
```

***tk_focusPrev***(self)

```
Return previous widget in the focus order. See tk_focusNext f
```

***tk_menuBar***(self, *args)

```
Do not use. Needed in Tk 3.6 and earlier.
```

***tk_setPalette***(self, *args, **kw)

```
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk
widget elements are derived from this.
Alternatively several keyword parameters and its associated
colors can be given. The following keywords are valid:
activeBackground, foreground, selectColor,
activeForeground, highlightBackground, selectBackground,
background, highlightColor, selectForeground,
disabledForeground, insertBackground, troughColor.
```

***tk_strictMotif***(self, boolean=None)

```
Set Tcl internal variable, whether the look and feel
should adhere to Motif.

A parameter of 1 means adhere to Motif (e.g. no color
change if mouse passes over slider).
Returns the set value.
```

***tkraise***(self, aboveThis=None)

```
Raise this widget in the stacking order.
```

***unbind***(self, sequence, funcid=None)

```
Unbind for this widget for event SEQUENCE  the
function identified with FUNCID.
```

***unbind_all***(self, sequence)

```
Unbind for all widgets for event SEQUENCE all functions.
```

***unbind_class***(self, className, sequence)

```
Unbind for a all widgets with bindtag CLASSNAME for event SEQ
all functions.
```

***update***(self)
```
Enter event loop until all pending events have been processed
```

***update_idletasks***(self)
```
Enter event loop until all idle callbacks have been called. T
will update the display of windows but not process events cau
the user.
```

***wait_variable***(self, name='PY_VAR')
```
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or
BooleanVar must be given.
```

***wait_visibility***(self, window=None)
```
Wait until the visibility of a WIDGET changes
(e.g. it appears).

If no parameter is given self is used.
```

***wait_window***(self, window=None)
```
Wait until a WIDGET is destroyed.

If no parameter is given self is used.
```

***waitvar*** = wait_variable(self, name='PY_VAR')
```
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or
BooleanVar must be given.
```

***winfo_atom***(self, name, displayof=0)
```
Return integer which represents atom NAME.
```

***winfo_atomname***(self, id, displayof=0)
```
Return name of atom with identifier ID.
```

***winfo_cells***(self)
```
Return number of cells in the colormap for this widget.
```

***winfo_children***(self)
```
Return a list of all widgets which are children of this widge
```

***winfo_class***(self)
```
Return window class name of this widget.
```

***winfo_colormapfull***(self)
```
Return true if at the last color request the colormap was ful
```

***winfo_containing***(self, rootX, rootY, displayof=0)
```
Return the widget which is at the root coordinates ROOTX, ROO
```

***winfo_depth***(self)

    Return the number of bits per pixel.

***winfo_exists***(self)

    Return true if this widget exists.

***winfo_fpixels***(self, number)

    Return the number of pixels for the given distance NUMBER
    (e.g. "3c") as float.

***winfo_geometry***(self)

    Return geometry string for this widget in the form "widthxhei

***winfo_height***(self)

    Return height of this widget.

***winfo_id***(self)

    Return identifier ID for this widget.

***winfo_interps***(self, displayof=0)

    Return the name of all Tcl interpreters for this display.

***winfo_ismapped***(self)

    Return true if this widget is mapped.

***winfo_manager***(self)

    Return the window mananger name for this widget.

***winfo_name***(self)

    Return the name of this widget.

***winfo_parent***(self)

    Return the name of the parent of this widget.

***winfo_pathname***(self, id, displayof=0)

    Return the pathname of the widget given by ID.

***winfo_pixels***(self, number)

    Rounded integer value of winfo_fpixels.

***winfo_pointerx***(self)

    Return the x coordinate of the pointer on the root window.

***winfo_pointerxy***(self)

    Return a tuple of x and y coordinates of the pointer on the r

***winfo_pointery***(self)

    Return the y coordinate of the pointer on the root window.

***winfo_reqheight***(self)

    Return requested height of this widget.

***winfo_reqwidth***(self)
```
    Return requested width of this widget.
```

***winfo_rgb***(self, color)
```
    Return tuple of decimal values for red, green, blue for
    COLOR in this widget.
```

***winfo_rootx***(self)
```
    Return x coordinate of upper left corner of this widget on th
    root window.
```

***winfo_rooty***(self)
```
    Return y coordinate of upper left corner of this widget on th
    root window.
```

***winfo_screen***(self)
```
    Return the screen name of this widget.
```

***winfo_screencells***(self)
```
    Return the number of the cells in the colormap of the screen
    of this widget.
```

***winfo_screendepth***(self)
```
    Return the number of bits per pixel of the root window of the
    screen of this widget.
```

***winfo_screenheight***(self)
```
    Return the number of pixels of the height of the screen of th
    in pixel.
```

***winfo_screenmmheight***(self)
```
    Return the number of pixels of the height of the screen of
    this widget in mm.
```

***winfo_screenmmwidth***(self)
```
    Return the number of pixels of the width of the screen of
    this widget in mm.
```

***winfo_screenvisual***(self)
```
    Return one of the strings directcolor, grayscale, pseudocolor
    staticcolor, staticgray, or truecolor for the default
    colormodel of this screen.
```

***winfo_screenwidth***(self)
```
    Return the number of pixels of the width of the screen of
    this widget in pixel.
```

***winfo_server***(self)
```
    Return information of the X-Server of the screen of this widg
    the form "XmajorRminor vendor vendorVersion".
```

***winfo_toplevel***(self)

```
    Return the toplevel widget of this widget.
```

***winfo_viewable***(self)

```
    Return true if the widget and all its higher ancestors are ma
```

***winfo_visual***(self)

```
    Return one of the strings directcolor, grayscale, pseudocolor
    staticcolor, staticgray, or truecolor for the
    colormodel of this widget.
```

***winfo_visualid***(self)

```
    Return the X identifier for the visual for this widget.
```

***winfo_visualsavailable***(self, includeids=0)

```
    Return a list of all visuals available for the screen
    of this widget.

    Each item in the list consists of a visual name (see winfo_vi
    depth and if INCLUDEIDS=1 is given also the X identifier.
```

***winfo_vrootheight***(self)

```
    Return the height of the virtual root window associated with
    widget in pixels. If there is no virtual root window return t
    height of the screen.
```

***winfo_vrootwidth***(self)

```
    Return the width of the virtual root window associated with t
    widget in pixel. If there is no virtual root window return th
    width of the screen.
```

***winfo_vrootx***(self)

```
    Return the x offset of the virtual root relative to the root
    window of the screen of this widget.
```

***winfo_vrooty***(self)

```
    Return the y offset of the virtual root relative to the root
    window of the screen of this widget.
```

***winfo_width***(self)

```
    Return the width of this widget.
```

***winfo_x***(self)

```
    Return the x coordinate of the upper left corner of this widg
    in the parent.
```

***winfo_y***(self)

```
    Return the y coordinate of the upper left corner of this widg
    in the parent.
```

---

Data and other attributes inherited from Tkinter.Misc:

***getdouble*** = <type 'float'>

```
float(x) -> floating point number

Convert a string or number to a floating point number, if pos
```

***getint*** = <type 'int'>

```
int(x[, base]) -> integer

Convert a string or number to an integer, if possible.  A flo
argument will be truncated towards zero (this does not includ
representation of a floating point number!)  When converting 
the optional base.  It is an error to supply a base when conv
non-string. If the argument is outside the integer range a lo
will be returned instead.
```

---

Methods inherited from Tkinter.Pack:

***forget*** = pack_forget(self)

```
Unmap this widget and do not use it for the packing order.
```

***info*** = pack_info(self)

```
Return information about the packing options
for this widget.
```

***pack*** = pack_configure(self, cnf={}, **kw)

```
Pack a widget in the parent widget. Use as options:
after=widget - pack it after you have packed widget
anchor=NSEW (or subset) - position widget according to
                          given direction
        before=widget - pack it before you will pack widget
expand=bool - expand widget if parent size grows
fill=NONE or X or Y or BOTH - fill widget if widget grows
in=master - use master to contain this widget
ipadx=amount - add internal padding in x direction
ipady=amount - add internal padding in y direction
padx=amount - add padding in x direction
pady=amount - add padding in y direction
side=TOP or BOTTOM or LEFT or RIGHT -  where to add this widg
```

***pack_configure***(self, cnf={}, **kw)

```
Pack a widget in the parent widget. Use as options:
after=widget - pack it after you have packed widget
anchor=NSEW (or subset) - position widget according to
                          given direction
        before=widget - pack it before you will pack widget
expand=bool - expand widget if parent size grows
fill=NONE or X or Y or BOTH - fill widget if widget grows
in=master - use master to contain this widget
ipadx=amount - add internal padding in x direction
ipady=amount - add internal padding in y direction
padx=amount - add padding in x direction
pady=amount - add padding in y direction
side=TOP or BOTTOM or LEFT or RIGHT -  where to add this widg
```

***pack_forget***(self)
```
    Unmap this widget and do not use it for the packing order.
```

***pack_info***(self)
```
    Return information about the packing options
    for this widget.
```

---

Methods inherited from Tkinter.Place:

***place*** = place_configure(self, cnf={}, **kw)
```
    Place a widget in the parent widget. Use as options:
    in=master - master relative to which the widget is placed.
    x=amount - locate anchor of this widget at position x of mast
    y=amount - locate anchor of this widget at position y of mast
    relx=amount - locate anchor of this widget between 0.0 and 1.
                  relative to width of master (1.0 is right edge)
        rely=amount - locate anchor of this widget between 0.0 ar
                  relative to height of master (1.0 is bottom edg
        anchor=NSEW (or subset) - position anchor according to gi
    width=amount - width of this widget in pixel
    height=amount - height of this widget in pixel
    relwidth=amount - width of this widget between 0.0 and 1.0
                      relative to width of master (1.0 is the sam
              as the master)
        relheight=amount - height of this widget between 0.0 and
                       relative to height of master (1.0 is the s
               height as the master)
        bordermode="inside" or "outside" - whether to take border
                                           into account
```

***place_configure***(self, cnf={}, **kw)

```
    Place a widget in the parent widget. Use as options:
    in=master - master relative to which the widget is placed.
    x=amount - locate anchor of this widget at position x of mast
    y=amount - locate anchor of this widget at position y of mast
    relx=amount - locate anchor of this widget between 0.0 and 1.
                  relative to width of master (1.0 is right edge)
        rely=amount - locate anchor of this widget between 0.0 ar
                  relative to height of master (1.0 is bottom edg
        anchor=NSEW (or subset) - position anchor according to gi
    width=amount - width of this widget in pixel
    height=amount - height of this widget in pixel
    relwidth=amount - width of this widget between 0.0 and 1.0
                      relative to width of master (1.0 is the sam
              as the master)
        relheight=amount - height of this widget between 0.0 and
                       relative to height of master (1.0 is the s
               height as the master)
        bordermode="inside" or "outside" - whether to take border
                                           into account
```

***place_forget***(self)

```
     Unmap this widget.
```

***place_info***(self)

```
     Return information about the placing options
     for this widget.
```

---

Methods inherited from Tkinter.Grid:

***grid*** = grid_configure(self, cnf={}, **kw)

```
     Position a widget in the parent widget in a grid. Use as opti
     column=number - use cell identified with given column (starti
     columnspan=number - this widget will span several columns
     in=master - use master to contain this widget
     ipadx=amount - add internal padding in x direction
     ipady=amount - add internal padding in y direction
     padx=amount - add padding in x direction
     pady=amount - add padding in y direction
     row=number - use cell identified with given row (starting wit
     rowspan=number - this widget will span several rows
     sticky=NSEW - if cell is larger on which sides will this
                   widget stick to the cell boundary
```

***grid_configure***(self, cnf={}, **kw)

```
     Position a widget in the parent widget in a grid. Use as opti
     column=number - use cell identified with given column (starti
     columnspan=number - this widget will span several columns
     in=master - use master to contain this widget
     ipadx=amount - add internal padding in x direction
     ipady=amount - add internal padding in y direction
     padx=amount - add padding in x direction
     pady=amount - add padding in y direction
     row=number - use cell identified with given row (starting wit
     rowspan=number - this widget will span several rows
     sticky=NSEW - if cell is larger on which sides will this
                   widget stick to the cell boundary
```

***grid_forget***(self)

```
     Unmap this widget.
```

***grid_info***(self)

```
     Return information about the options
     for positioning this widget in a grid.
```

***grid_remove***(self)

```
     Unmap this widget but remember the grid options.
```

***location*** = grid_location(self, x, y)

```
     Return a tuple of column and row which identify the cell
     at which the pixel at position X and Y inside the master
     widget is located.
```

class ***create***

```
#----------------------------------------------------------------------------
#
# Start of Popup Dialog
#
#----------------------------------------------------------------------------
# VCS Plot Annotation Dialog Popup
#----------------------------------------------------------------------------
```

Methods defined here:

***__init__***(self, parent)

***annotate_cancel***(self, parent)

***annotate_clear***(self, parent)

***annotate_replot***(self, parent)

***annotate_reset***(self, parent)

***evt_set_toggle_state***(self, parent, id)

***execute***(self, parent, result)

***get_settings***(self, parent)

***hold_annotate_cancel_settings***(self, parent)

***hold_annotate_original_settings***(self, parent)

***master_switch***(self, parent, result)

***retain_switch***(self, parent, result)

## *Functions*

***get_annotation_info***(parent, data_name=None)

```
#----------------------------------------------------------------------------
#
# Get the annotation infomation from the data variable
#
#----------------------------------------------------------------------------
```

## *Data*

***Pmw*** = <Pmw.Pmw_1_2.lib.PmwLoader.PmwLoader instance>